**Лабораторная работа №5**

**Тема**: Геометрические преобразования на плоскости

**Цель работы:** Научиться выполнять геометрические преобразования над

плоскими фигурами.

**Теоретический материал.**

Матриці основних елементарних перетворень:

* матриця повороту(rotation) точки відносно початку координат у додатному напрямку:



* матриця розтягу(стиску) (dilation) відносно початку координат:



* матриця перенесення(translation) точки на вектор(m, n):



* матриця дзеркального відображення(reflection) відносно осі x:



Перетворення системи координат задається матрицями:

* *R(–φ)* – поворот системи координат на кут φ у додатному напрямі;
* *T(–m, –n)* – зсув системи координат на вектор (*m, n*).

Зауважимо, що для простоти запису аргументи наведених матриць іноді опускаються.

Розглянемо декілька практичних прикладів.

***Приклад 5.1***. Побудувати матрицю повороту точкиM( x, y) відносно довільної точкиN(m, n) на кут φ у додатному напрямку (рис. 5.1).

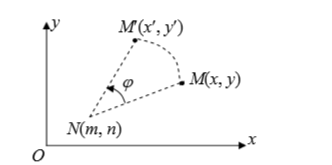


Рис. 5. Поворот точки M у додатному напрямку відносно точки N

Матриця R задає поворот точки відносно початку координат. Однорідні координати дають можливість знайти матрицю повороту відносно довільної точки. У загальному випадку поворот відносно довільної точки може бути реалізований шляхом таких перетворень:

1. переміщення точки *N(m, n)* на вектор(–m, –n) так, щоб центр повороту сумістився з початком координат. Матриця цього перетворення



1. повороту точки на кут φ у додатному напрямку відносно початку координат. Матриця цього перетворення R визначається формулою



1. переміщення одержаного результату назад так, щоб центр повороту сумістився з точкою *N*. Матриця цього перетворення T визначається формулою



Отже, для знаходження результуючого повороту точки *M(x, y)* відносно точки *N(m, n)* потрібно перемножити матриці T’, R, T за вказаним порядком.

У результаті одержимо шукане перетворення у вигляді



Зокрема, поворот точки на900 у додатному напрямку відносно точки *N(m, n)* виконується перетворенням

.

***Приклад 2***. Побудувати матрицю неоднорідного масштабування з коефіцієнтом *a* вздовж осі *x* та коефіцієнтом *b* уздовж осі *y* з центром у точці *N(m, n)*.

Це перетворення реалізуємо за допомогою трьох перетворень:

1. переміщення точки на вектор *N(–m, –n)*. Матрицею цього перетворення є матриця *T’*.
2. розтягу вздовж координатних осей із коефіцієнтами *a* та *b* відповідно. Матриця *D* цього перетворення визначається формулою

;

1. перенесення одержаного результату на вектор *N(m, n).* Матриця *T* цього перетворення має вигляд.



Перемножуючи ці матриці, одержуємо шукане перетворення



Аналогічно, виділяючи послідовність операцій перенесення, повороту та масштабування(ці перетворення описуються матрицями *T, R, D*), можна побудувати матрицю для будь-якого складного афінного перетворення. Тоді афінне перетворення об’єктів на площині можна здійснити так: якщо об’єкт задається сукупністю точок, то складаємо матрицю *X* однорідних координат точок цього об’єкта і цю матрицю множимо на матрицю складного афінного перетворення.

Розглянемо приклад симетричного відображення трикутника відносно прямої *L*.

***Приклад 3***. Нехай пряма *L* задається рівнянням *y = x + 1*, а однорідні координати *A(1; 1; 1), B(6; 1; 1),C (6; 4; 1)* задають координати вершин трикутника *ABC*. Побудувати трикутник, симетричний даному відносно прямої *L*.

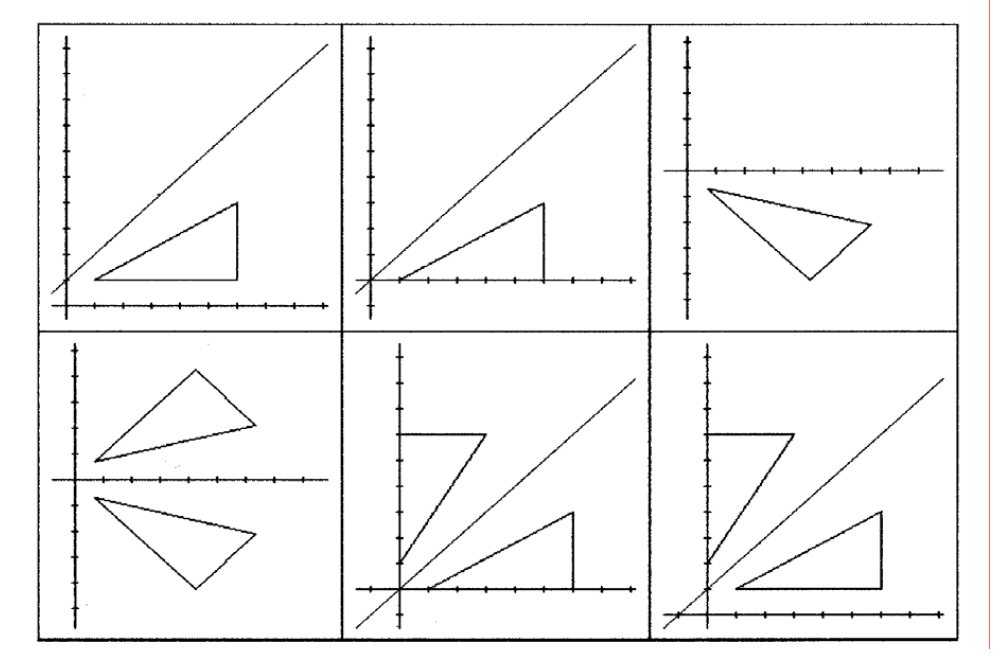


Рис. 5. Побудова симетричного трикутника

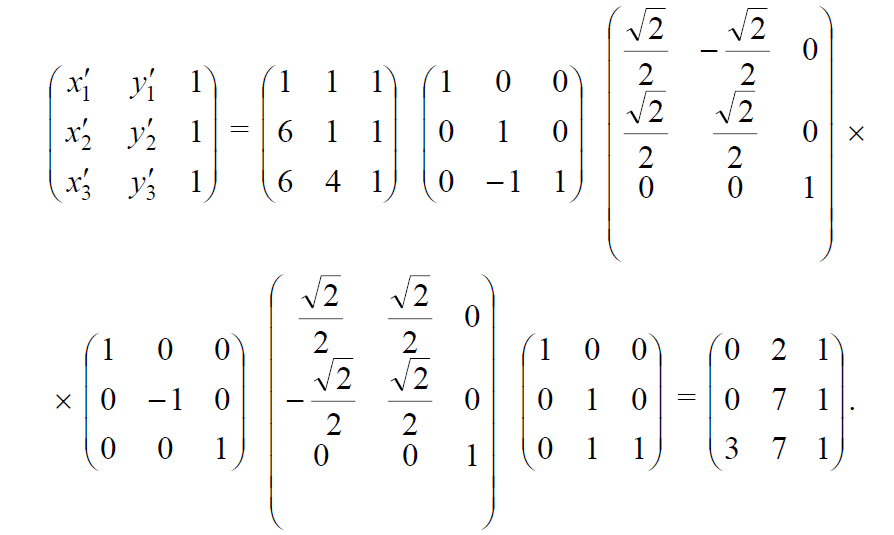
Представимо перетворення симетрії точки відносно прямої у вигляді послідовності простіших афінних перетворень (рис. 5.2):

* переміщення лінії та трикутника так, щоб лінія *L* пройшла через початок координат. Це перетворення задає матриця *T* із параметрами(0, –1);
* поворот лінії та об’єкта відносно початку координат до збігу лінії *L* з віссю x. Це перетворення задає матриця R із кутом  (у від’ємному напрямку);
* відображення відносно координатної осі x;
* обернений поворот відносно початку координат;
* паралельне перенесення у вихідне положення.

Останні три перетворення задаються матрицями *Ref,* *R(-φ*) та *T(0, 1)*, відповідно. У матричному вигляді перетворення симетрії має вигляд



Для симетрії трикутника *ABC* відносно прямої *L* маємо



Отже, декартові координати вершин симетричного трикутника *A′B′C′* мають значення A′(0; 2), B′(0; 7), C′(3; 7).

**Задания.**

**Для решения задач можно использовать примитивы**

***1 уровень***

Данная работа содержит варианты разного уровня, поэтому прежде чем делать задание, выясните у Вашего преподавателя ваш вариант.

Фигуру, подвергающуюся преобразованию, задайте массивом векторов, где каждый вектор представляет координаты одной из вершин фигуры.

Необходимое преобразование задавайте матрицей3х3. Матрицу преобразования, являющегося комбинацией нескольких преобразований, формируйте путем последовательных перемножений матриц соответствующих преобразований.

Следует программировать задачу в общем виде, используя параметры(угол поворота, величину перемещения, коэффициент масштабирования) и предполагая произвольную фигуру, но отлаживать программу на примере, указанном в задаче в скобках!

На рисунках, приведенных в вариантах заданий, жирными линиями изображено первоначальное положение фигуры.

Включив в программу необходимые задержки(используйте таймер), добейтесь желаемой скорости смены – изображений.

|  |  |
| --- | --- |
| Вариант | Рисунок |
|  | Поворачивать любую фигуру, образованную замкнутой ломаной линией (секундную стрелку), вокруг одной из ее вершин (оси вращения) на небольшой угол *r* по часовой стрелке (соответствующий секунде). Совершить *n* поворотов.    Рис. 5. Секундная стрелка  ***Дополнение:*** На основе полученной программы создайте часы - будильник, звонящие по достижении заданного времени. К рисунку добавьте часовую, минутную стрелки и стрелку будильника. |
|  | Поворачивать любую фигуру, образованную отрезками прямых (велосипедное колесо ), и расположенную у левого края экрана, вокруг заданной точки фигуры (оси колеса), на угол *r* по часовой стрелке (на половину угла между соседними спицам), сдвигая, фигуру, по горизонтали вправо на величину *t*. Фигуру перемещать до правого края экрана. Один, из отрезков (спицу колеса) изобразить контрастным цветом. Величину *t* рассчитайте так, чтобы создавалось впечатление катящегося колеса, и проведите пунктирную горизонталь, через точки вращения.    Рис. 5. Колесо  ***Дополнение:*** Создайте картину движущегося по экрану велосипеда. |
|  | Задана любая фигура двумя замкнутыми ломаными линиями (треугольник внутри квадрата). Требуется поворачивать фигуру вокруг заданной точки *z* (центра квадрата) на угол *r* (15 градусов) по часовой стрелке, незначительно уменьшая относительно точки и часть фигуры, образованную первой ломаной линией(треугольник), и во столько же раз увеличивая другую часть фигуры(квадрат). Преобразования повторять, пока уменьшающаяся часть не превратится в точку. Картина должна напоминать"вертушку", состоящую из удаляющегося при вращении треугольника и приближающегося квадрата. Затем повторить процесс, только уменьшать вторую часть фигуры и увеличивать первую. Повторять чередование, пока пользователь не завершит работу программы.    Рис. 5. Вертушка  ***Дополнение:*** Изменяйте скорость поворота маятника в соответствии с гармоническим закон колебаний. Создайте программу часов с маятником. |
|  | Поворачивать любую фигуру, образованную отрезками прямых (маятник  в крайнем левом положении, отклоненный, на 45 градусов от вертикали),  вокруг точки в центре верхней половины экрана(оси маятника) на небольшой угол r (5 градусов) против часовой стрелки. Совершить k поворотов(достигнув крайнего правого положения). Затем, аналогично, выполнить k поворотов по часовой стрелке(до крайнего левого положения). Выполнить колебания фигуры, пока пользователь не завершит работу программы. Процесс должен напоминать колебание маятника    Рис. 5. Маятник |
|  | Поворачивать любую фигуру, образованную отрезками прямых (лист  дерева) и находящуюся в правой половине экрана, вокруг точки Р, лежащей в центре верхнего края экрана, на угол r (10 градусов) по часовой стрелке, каждый раз незначительно увеличивая фигуру относительно точки P. Повороты выполнять до тех пор, пока фигура не займет примерно симметричное первоначальному положение в левой половине экрана. Затем, аналогично, поворачивать фигуру против часовой стрелки вокруг той же точки P. Чередовать такие колебания фигуры, пока она остается в пределах экрана. Картина должна напоминать падение осеннего листа, сорвавшегося с дерева    Рис. 5. Падение листа  ***Дополнение:*** Создайте полную картину вращающегося колеса обозрения с шестью кабинами. |
|  | Задана любая фигура, состоящая из отрезков(треугольник и прямоугольник с общей стороной, напоминающие кабину колеса обозрения); одна из точек фигуры особо выделена(точка подвески кабины). Поворачивать фигуру так, чтобы особая точка поворачивалась вокруг центра экрана на угол r (15 градусов) против часовой стрелки при каждом повороте. Совершать полные обороты фигуры вокруг центра экрана, пока пользователь не завершит работу программы. Картина должна напоминать вращение кабины колеса обозрения.    Рис. 5. Колесо обозрения  ***Дополнение:*** Создайте полную картину вращающегося колеса обозрения с шестью кабинами. |
|  | Поворачивать любую фигуру, заданную замкнутой ломаной линией с яркой точкой на фигуре(многоугольник, похожий на круг, с ярким"пятном" на  окружности), вокруг центра фигуры на угол r (30 градусов) по часовой стрелке, сдвигая фигуру так, чтобы она перемещалась по диагонали экрана, и  незначительно увеличивая размеры фигуры в а раз (1.2 раза) относительно своего центра, но так, чтобы пятно, вращаясь вместе с фигурой, оставалось на прежнем расстоянии от ее центра. Картина должна напоминать катящийся с горы нарастающий снежный ком, а пятно напоминать о первоначальных размерах кома. Нарисовать гору и остановить ком в углу экрана.    Рис. 5. Снежный ком |
|  | Задана любая фигура замкнутой ломаной линией(квадрат). Поворачивать фигуру вокруг своего центра на угол r (35 градусов) против часовой стрелки и одновременно вокруг центра экрана на угол s (20 градусов) по часовой стрелке. Одну из сторон фигуры выделить другим цветом. Совершать полные обороты фигуры вокруг центра экрана, пока пользователь не завершит работу программы. Картина должна напоминать карусель с вращающейся кабиной.    Рис. 5. Карусель с вращающейся кабиной  ***Дополнение:*** Создайте полную картину карусели с четырьмя вращающимися кабинами. |
|  | Поворачивать любую фигуру, заданную замкнутой ломаной линией (вытянутый прямоугольник, стоящий у левого края экрана на наклонной линии), на угол r (90 градусов) по часовой стрелке вокруг первой вершины фигуры, затем на тот же угол вокруг второй вершины и т. д. k раз(каждый раз в качестве точки поворота выбирать нижний правый угол очередного прямоугольника, Одну из сторон фигуры(короткую) выделить другим цветом.    Рис. 5. Кувырок  ***Дополнение:*** Добейтесь того, чтобы прямоугольник стоял на выделенной стороне дольше, чем на остальных, и докувыркался до правого края экрана. |

***2-уровень (максимальное количество баллов 80) выполнение всех пунктов задания.***

Создать систему декартовых координат с сеткой.

В составе системы координат изобразить геометрическую фигуру, согласно варианту задания из таблицы 1. Фигура должна задаваться координатами вершин: A(x1,y1), B(x2,y2), C(x3,y3), и D(x4,y4) таким образом, чтобы она находилась в первом квадранте декартовой системы координат XOY, как показано на рисунке 6.1

Задавая различные значения параметров видовых преобразований продемонстрировать следующие виды графических преобразований:

1) Перенос фигуры из 1-го во 2-й, 3-й и 4-й квадранты;

2) Сдвиг объекта вдоль каждой оси координат или вдоль обеих осей одновременно;

3) Изменение масштаба объекта по оси X или по оси Y, а также изменение его размера по обеим осям одновременно;

Преобразования типа "перенос" и "масштабирование" должны выполняться в режиме анимации. При этом необходимо принять меры к устранению эффекта «мерцания» изображения, возникающего в процессе этих преобразований.

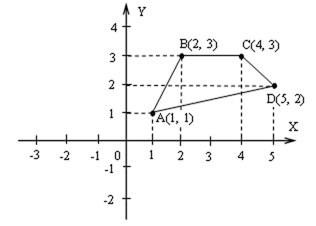


Рис. 5. Система координат и исходная фигура

Для решения задачи можно построить и затем воспользоваться формой

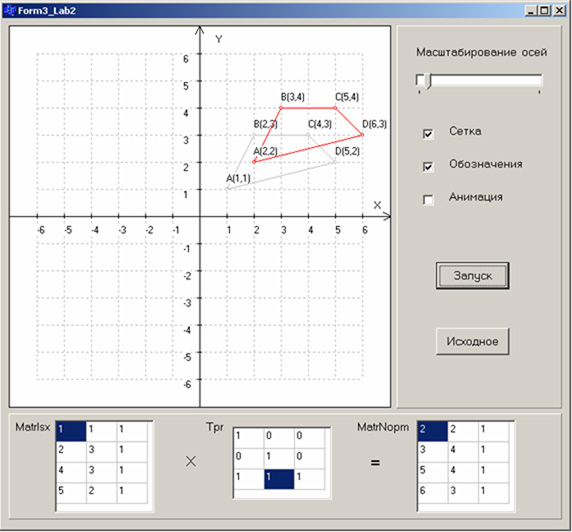


Рис. 5. Окно формы

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | квадрат | красный | черный | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | прямоугольник | синий | голубой | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | параллелограм | зеленый | коричневый | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | ромб | голубой | синий | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | треугольник | красный | черный | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | квадрат | красный | коричневый | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | прямоугольник | синий | голубой | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | параллелограмм | зеленый | черный | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | трапеция | коричневый | синий | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | ромб | голубой | зеленый | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | произвольный четырехугольник | черный | синий | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | равносторонний треугольник | зеленый | синий | |
|  | |  |  |  | | --- | --- | --- | | Тип фигуры | Цвет фигуры | Цвет осей | | Прямоугольный треугольник | синий | Зеленый | |

Реализация задания второго уровня.

1. ***Установка панели Panel1 на форму.***

В окне **ToolBox** палитры компонентов выберите значок **Panel**, и установите панель на поле **Form1**, разверните панель, используя растягивание указателем мыши до требуемых размеров , панель должна быть вертикальной;

Привяжите панель **Panel1** к верхнему, правому и нижнему краю формы, для чего в окне **Properties** переустановите свойство **Anchors** панели в состояние **[akTop, akRight, akBottom].**

В окне **Properties**, в строке **Text** удалите запись **Panel1**.

1. ***Установка панели Panel2 на форму.***

Измените размеры **Form3** в высоту на величину высоты панели **Panel2** .

Установите вторую панель **Panel2**  так, как показано на рисунке 5.13 способом, аналогичным п.1).

Привяжите панель **Panel2** к левому, правому и нижнему краю формы, для чего в окне **Properties** переустановите свойство **Anchors** панели в состояние **[akLeft, akRight, akBottom].**

В окне **Properties**, в строке **Text** удалите запись **Panel2**.

1. ***Установка компонента TrackBar1 ("Масштабирование осей") на панель Panel1.***

На вкладке **AllWindowsForm** выберите компонент **TrackBar**  и установите его на поле панели **Panel1**.

Для компонента **TrackBar1** в окне **Properties** выполните следующие настройки:

- в строке**Max** (максимальное количество отсчетов) установите значение: 20;

- в строке**Min** (минимальное количество отсчетов) установите значение:  5;

- в строке **Value** (начальная позиция движка) установите значение: 10;

- в строке**TickStyle** установите значение: **tsManual**;

1. ***Установка компонентов CheckBox на панель управления  Panel1.***

В группе A**llWindowsForm** выберите компонент **CheckBox**  и установите его на поле панели **Panel1**.

В окне **Properties**, в строке **Text** удалите запись **CheckBox1** и введите необходимый текста в соответсвтии с рисунком. Всего, согласно рисунку 5.13, надо установить три таких компонента с соответствующими подписями (сетка, обозначения, анимация).

Введите надписи "**Масштабирование осей**", "**Сетка**", "**Обозначения**" и "**Анимация**" на поле панели **Panel1** в соответствии с рисунком.

1. ***Установка кнопок "Запуск" и "Исходное" на панель управления.***

В окне **ToolBox** щелкните по значку **Button**, а затем щелкните внутри области **Panel1** поля **Form1**, возникнет кнопка **Button1**, которую можно переместить в требуемое место панели и изменить её размеры.

В окне **Properties**, в строке **Caption** вместо записи **Button1**  введите:  **Пуск**.

Аналогично выполнить действия по созданию кнопки **Исходное**.

1. ***Установка компонентов DataGridView1 на панель управления  Panel2.***

В окне **ToolBox** выберите компонент **DataGridView** и установите его на поле панели **Panel2**.

Для компонента **DataGridView1** в окне **Properties** выполните следующие настройки:

- в строке **RowHeadersVisible** (отображение заголовков строк) установите значение **False**;

- в строке **ColumnHeadersVisible** (отображение заголовков столбцов) установите значение **False**;

- в строке **ScrollBars** (отображение полос прокрутки) установите значение **None (нет)**;

- в метод **InitializeComponent()** в области, связанной с **DataGridView1** введите следующую команду

this->dataGridView1->ColumnCount=3;(определяем количество столбцов в массиве 3)

- в строке **AvtoSizeColumnMode** (занимать пространство компонента) установите значение **Fill**;

- в строке **GridColor** (цвет сетки) установите значение **ActiveCaptionText** (черный);

- в строке **AllowUserToAddRows**  (фиксируем количество строк) установите значение **false**.

Щелкните по компоненту **DataGridView1**правой кнопкой мыши и в открывшемся меню выберите **Copy**. Щелкните правой кнопкой на поле панели **Panel2** и в меню выберите **Paste**. На панели будет создана копия компонента - **DataGridView2**. Аналогичным образом создайте третий компонент - **DataGridView3.** Разместите все три компонента **DataGridView** на панели **Panel2** согласно рисунку 5.12.

Для компонента **DataGridView2** исправьте свойство **RowCount**  на значение:  3, т.е явно пропишите this->dataGridView2->RowCount=3;

Для компонента **DataGridView3** явно пропишите this->dataGridView3->RowCount=4.

1. ***Установка компонентов Label на панель управления  Panel2:***

Используя компоненты Label и их свойство text, введите математические символы "**Х**" и "**=**" на поле панели **Panel2**в соответствии с рисунком 5.12.

Установите названия матриц для компонентов **DataGridView**  на панели **Panel2** согласно рисунку 5.12.

По требованию задачи проект должен выполнять следующие действия:

- создание осей координат                                               – **ShowOsi()**;

- изображение сетки на поле графического окна       – **ShowStk()**;

- вывод матриц исходных и результирующих координат и матрицы преобразования;

- масштабирование осей координат и изображения в них;

- видовые преобразования исходной фигуры             – **Preobr()**;

- нормировка координат преобразованной фигуры – **Normir()**;

- прорисовка исходной и преобразованной фигуры –

**ShowFig(TColor col, float matrT[4][3])**;

- прорисовка обозначений вершин фигуры – **Oboznath(float matrTo[4][3])**;

- установка фигуры в исходное положение.

Запуск всех функций прорисовки объединим под одной общей функцией - **ShowAll()**, а запуск всех функций преобразования и прорисовки изображения объединим под функцией - **GrafKonv()**, которая будет выполнять роль графического конвейера.

Для таких действий как вывод матриц исходных и результирующих координат, масштабирование осей координат и установка фигуры в исходное положение отдельные функции создавать не будем, для их реализации достаточно использовать стандартные обработчики **DataGridView**, **TrackBar1** и **Button2**.

Запуск функции **GrafKonv()**, будем осуществлять из обработчика **FormPaint()**, который должен вызываться при каждом обращении пользователя к элементам управления (кнопкам. движкам, флажкам и т.п.). Порядок программирования этих обработчиков будет рассмотрен ниже.

Объектами обработки для функций **Preobr()**, **Normir()**, **ShowFig()** и **Oboznath()** являются координаты фигуры **ABCD**, сгруппированные в матрицы. Координаты находятся на различных стадиях обработки, и поэтому они должны быть помещены в отдельные матрицы с различными именами, соответствующими этим стадиям, а именно:

matrIsx[4][3] - матрица исходных координат;

matrIzm[4][3] - матрица преобразованных (измененных) координат;

matrRez[4][3] - матрица ортонормированных координат;

matrEkr[4][2] - матрица экранных координат.

Матрицу исходных координат мы создаем для того, чтобы обеспечить первоначальную прорисовку фигуры, сразу после запуска проекта или после нажатия на кнопку **"Исходное"**.

Матрица измененных координат получается после преобразования исходных координат фигуры с помощью матрицы видового преобразования **Tpr[3][3]**.

Матрицу результирующих координат мы получим после нормировки преобразованных координат.

Матрицу экранных координат получим после пересчета ортонормированных координат в экранные.

Кроме этого, для преобразования координат фигуры необходима матрица видового преобразования - **Tpr[3][3]**.

Поэтому, с учетом вышесказанного, в начальной части файла реализации объявим глобально все матрицы координат фигуры, матрицу преобразования фигуры и прототипы всех используемых функций преобразования и прорисовки изображения, а также все наиболее употребимые переменные в следующем виде:

Graphics::TBitmap \* formBitmap;          // Описание указателя на буфер

static int W, H, nOs, h, hh, i, j, k;

static float a, b, d, e, l, m, p, q, s;

static float matrIsx[4][3];

static float matrIzm[4][3];

static float matrRez[4][3];

int matrEkr[4][2];

static float Tpr[3][3];

static void GrafKonv(void);

static void Preobr(void);

static void Normir(void);

static void ShowAll(void);

static void ShowOsi(void);

static void ShowStk(void);

static void ShowFig(TColor col, float matrT[4][3]);

static void Oboznath(float matrTo[4][3]);

Здесь, как видим, объявлен также ссылка на графический буфер класса **TBitmap**.

Графический буфер – это битовая матрица, служащая для хранения в ней изображения (картинки) в формате **.bmp**, т.е. в пикселях.

Необходимость создания буфера в этой лабораторной работе определяется тем обстоятельством, что видовые преобразования фигуры должны выполняться в режиме анимации, осуществление которой потребует многократной перерисовки изображения. Если не применять отрисовку с буферизацией, то качество изображения будет очень низким из-за появления черных полос в области графического окна проекта.

Порядок использования графического буфера следующий:

- сначала забелим поверхность битмапа, т.е. графического буфера;

- после этого на его поле можно выполнять прорисовки с использованием команды:

formBitmap->Canvas-> <имя функции прорисовки>;

- затем содержимое буфера копируется в графическое окно формы с помощью команды:

Form3->Canvas->Draw(0, 0, formBitmap);

Здесь, параметр **formBitmap** - это имя буфера, его можно выбирать любым. Практический пример использования буфера рассмотрим далее, при описании функции **FormPaint()**.

**Разработка функции инициализации исходных данных Form1\_Load()**

**FormCreate()**

Обработчик события **Form1\_Load()** позволяет выполнять операцию предварительной инициализации данных сразу при запуске программы. Для его создания надо открыть вкладку **Events** окна **Properties** для формы **Form1** и щелкнуть по строке **Load**, затем дважды щелкнуть по открывшемуся белому окошку. В окне редактора кода появится заголовок новой функции:

**private: System::Void Form1\_Load(System::Object^ sender, System::EventArgs^ e) {}**

void \_\_fastcall TForm1::FormCreate(TObject \*Sender){}

Внутри фигурных скобок этой функции впишем команды обеспечивающие следующее:

- инициализацию графического буфера;

- инициализацию матрицы исходных координат **matrIsx[4][3]**;

- инициализацию матрицы видового преобразования **Tpr[3][3]**;

- инициализацию переменной **nOs**.

Для инициализации графического буфера впишем следующие команды:

W=this->Width - groupBox1->Width -10; // Задаем ширину графич. окна на форме

H=this->Height - groupBox2->Height-10; // Задаем высоту графич. окна на форме

formBitmap=new Graphics::TBitmap();                      // Выделям динамич. память под буфер

formBitmap->Width = W;                                            // Задаем ширину битмапа

formBitmap->Height= H;                                              // Задаем высоту битмапа

formBitmap->Canvas->Brush->Color=clWhite;       // Задаем цвет закраски битмапа

Инициализацию матрицы исходных координат выполним следующим образом:

matrIsx[0][0]=1;   matrIsx[0][1]=1;   matrIsx[0][2]=1;

matrIsx[1][0]=2;   matrIsx[1][1]=3;   matrIsx[1][2]=1;

matrIsx[2][0]=4;   matrIsx[2][1]=3;   matrIsx[2][2]=1;

matrIsx[3][0]=5;   matrIsx[3][1]=2;   matrIsx[3][2]=1;

Понятно, что каждый студент здесь должен вписать другие численные значения. Координаты вершин фигуры можно выбирать любые, но так, чтобы форма фигуры соответствовала заданному варианту.

Далее, инициализируем матрицу видового преобразования. В начальный период она имеет вид единичной матрицы, так как ни один из её параметров еще не "включен":

Tpr[0][0]=1;   Tpr[0][1]=0;   Tpr[0][2]=0;

Tpr[1][0]=0;   Tpr[1][1]=1;   Tpr[1][2]=0;

Tpr[2][0]=0;   Tpr[2][1]=0;   Tpr[2][2]=1;

Затем выводим значения элементов матриц **matrIsx[4][3]** и **Tpr[3][3]** в соответствующие окна вывода **DataGridView1** и **DataGridView2**. Пример заполнения первых двух строк для каждого из этих окон имеет вид:

for(int j=0; j<dataGridView1->RowCount;j++)

for (int i=0;i<dataGridView1->ColumnCount;i++)

dataGridView1->Rows[j]->Cells[i]->Value=matrIsx[j][i].ToString();

for(int j=0; j<dataGridView2->RowCount;j++)

for (int i=0;i<dataGridView2->ColumnCount;i++)

dataGridView2->Rows[j]->Cells[i]->Value=Tpr[j][i].ToString();

В заключение зададим количество отсчетов на осях **X** и **Y** с помощью переменной: **nOs** = 10.

**7.4. Создание  обработчика события  FormPaint**

В отличие от лабораторной работы №1, где мы выполняли прорисовку изображения на поле объекта **PaintBox1,** в этой лабораторной работе предлагается выполнить прорисовку прямо на поле самой формы **Form3,** используя свойство **Canvas**, которым обладает любая форма. Для этого откройте вкладку **Events** окна **Object Inspector**и щелкните по строке**OnPaint**. Для создания обработчика события  **FormPaint**  дважды щелкните по открывшемуся белому окошку.

Перейдите в окно **Unit3.cpp** с текстом программы. В нем появилась новая функция  **FormPaint,** внутри фигурных скобок которой надо вписать команды обработки графического буфера. После чего текст программного кода функции будет иметь следующий вид:

void \_\_fastcall TForm3::FormPaint(TObject \*Sender)

{

 formBitmap->Canvas->FillRect(Rect(0, 0, W, H)); // Забеливаем битмап

 GrafKonv();                                                               // Вызываем конвейер (рисуем на битмапе)

 Canvas->Draw(0, 0, formBitmap);                           // Перерисовываем битмап на форму

}

Из текста комментариев к командам понятны действия, выполняемые данным обработчиком.

Первая команда подготавливает поле графического буфера для рисования.

Вторая команда выполняет все необходимые прорисовки на битмапе (см. п. 7.5).

Третья команда копирует изображение битмапа в ту область формы, которая задана параметрами канвы.

Закомментируйте на период отладки вызов функции **GrafKonv()** и запустите проект на исполнение. Убедитесь, что в окне проекта формы **Form3,** в его левой части, появился белый прямоугольник графического окна.

**7.5. Разработка функции графического конвейера GrafKonv()**

В задачу функции **GrafKonv()** входит последовательный запуск функций, ответственных за видовые преобразования фигуры, нормировку преобразованных координат фигуры и полную прорисовку всего изображения. Текст функции будет иметь вид:

void GrafKonv(void)

{

  Preobr();           // Вызываем функцию преобразования фигуры

  Normir();          // Вызываем функцию нормировки преобразованных координат

  ShowAll();        // Рисуем на битмапе всё: сетку, оси, фигуру, обозначения

}

Если запустить на этом этапе программу, то компилятор выведет сообщение об ошибке, т.к. ни одна из функций, вызываемых здесь, еще не существует, их описание будем разрабатывать далее. А пока, чтобы можно было продолжать формирование текста программы и выполнять отладку её функций, надо закомментировать, т.е. заглушить вызовы не созданных функций **Preobr()**,**Normir()** и **ShowAll()**. Затем раскомментируйте вызов **GrafKonv()** в обработчике **FormPaint** и запустите программу на исполнение. После добавления в текст программы описания каждой из вызываемых функций, заглушки можно будет последовательно снимать.

**7.6. Видовые преобразования координат фигуры. Разработка функции Preobr()**

Для изменения местоположения, формы или размеров фигуры к ней необходимо применить видовые преобразования. Для преобразования матрицы координат фигуры с помощью матрицы видового преобразования создадим функцию  **Preobr()**. В её работу будет входить выполнение операции перемножения двух матриц: матрицы исходных координат **matrIsx[4][3]** и матрицы видового преобразования **Tpr[3][3]**. В результате этой операции мы получим преобразованную матрицу **matrIzm[4][3]** (в соответствии с выражением (1), представленным в разделе "Краткие теоретические сведения"). Эту матрицу затем необходимо отнормировать с помощью функции **Normir()**. Запуск функции преобразования **Preobr(),** а также функции нормировки фигуры осуществляется после нажатия на кнопку "**Запуск**". При этом вызывается процесс перерисовки графического окна и соответственно запуск графического конвейера **GrafKonv()**, в составе которого и находятся функции **Preobr()** и **Normir()**.

Текст программного кода функции **Preobr()** имеет следующий вид:

void Preobr(void)

{

  for(i=0; i<4; i++)

  {for(j=0; j<3; j++)

     {float Sum=0;

       for(k=0; k<3; k++)

         Sum=Sum+matrIsx[i][k]\* Tpr[k][j];

       matrIzm[i][j]=Sum;

      }

   }

}

Здесь как видим выполняется поэлементное перемножение двух матриц в соответствии с правилами перемножения матриц.

Раскомментируйте вызов функции **Preobr()**, находящийся в **GrafKonv()** и запустите проект на исполнение для обнаружения ошибок в программном коде.

**7.7. Нормировка координат фигуры**

Нормировку преобразованных координат фигуры, записанных в матрице **matrIzm[4][3]** необходимо выполнять для перевода этих координат из однородной системы координат в декартовую. Принцип нормировки координат результирующей матрицы определяется математическим выражением (5), представленным в разделе "Краткие теоретические сведения", данной лабораторной работы.

Создадим функцию **Normir()**, внутри которой организуем два вложенных цикла, необходимых для обработки матрицы координат фигуры и с их помощью выполним поэлементное деление параметров матрицы на последний элемент текущей строки, если он не равен единице. Кроме этого, надо проверить, не равен ли этот последний элемент нулю, потому что, понятно, на нуль делить нельзя. Если все-таки это случилось, то этому нулевому элементу надо присвоить какое-либо очень маленькое значение.

void Normir(void)

{

  for(int i=0; i<4; i++)

     for(int j=0; j<3; j++)

       {if(matrIzm[i][2]!=1)

           {if(matrIzm[i][2]==0)

                 matrIzm[i][2]=0,001;

             matrRez[i][j]=matrIzm[i][j]/matrIzm[i][2];

            }

          else matrRez[i][j]=matrIzm[i][j];

      }

}

Отнормированные координаты преобразованной фигуры **matrRez[4][3]** можно теперь использовать в функциях **ShowFig()** и **Oboznath()**  для прорисовки фигуры и простановки обозначений возле её вершин.

Раскомментируйте вызов функции **Normir()**, находящийся в **GrafKonv()** и запустите проект на исполнение для проверки.

**7.8. Разработка функции прорисовки  изображения ShowAll()**

В задачу функции **ShowAll()** входит последовательный запуск функций, ответственных за прорисовку конкретных элементов изображения: осей координат, сетки, фигуры и обозначения вершин фигуры. В обязанность этой функции также входит проверка состояния флажков "**Сетка**" и "**Обозначения**", установленных на панели **Panel1**.

 Порядок вызова функций прорисовки, входящих в состав **ShowAll()**, определяется тем, что каждое последующее изображение накладывается на предыдущее. Поэтому, чтобы не происходила зарисовка важной информации нужно определить, какое изображение является основным, а какое – второстепенным или фоновым. В нашем проекте основным является преобразованное изображение фигуры, а исходная фигура, оси координат, сетка, обозначения вершин – второстепенными. Поэтому, сначала надо вызывать второстепенные функции в порядке их нарастающей значимости, а затем основную. Тогда текст функции **ShowAll()** будет иметь вид:

void ShowAll(void)

{

 if(Form3->CheckBox1->Checked)    // Если флажок "Сетка" включен,

    ShowStk();                                      // рисуем сетку

 ShowOsi();                                        // Рисуем оси

 ShowFig(clSilver, matrIsx);               // Рисуем исходную фигуру

 if(Form3->CheckBox2->Checked)    // Если флажок "Обозначения"включен,

    Oboznath(matrIsx);                         // проставляем исходные обозначения

 ShowFig(clRed, matrRez);                 // Рисуем преобразованную фигуру

 if(Form3->CheckBox2->Checked)    // Если флажок "Обозначения" включен,

    Oboznath(matrRez);                       // проставляем преобразованные обозначения

}

Как видно из текста программы, функции прорисовки фигуры и обозначений вершин вызываем дважды: первый раз рисуется исходная фигура серым цветом, второй раз рисуется преобразованная фигура красным цветом.

Пока не созданы описания функций **ShowStk()**, **ShowOsi(),** **ShowFig()** и **Oboznath()**   закомментируйте их вызовы в **ShowAll()**. Затем раскомментируйте вызов **ShowAll()** в функции **GrafKonv()** и запустите проект на исполнение для проверки.

**7.9. Разработка функции построения осей координат ShowOsi()**

Процесс создания функции **ShowOsi()** следующий:

а) В программный код файла реализации **Unit3.cpp** впишите заголовок этой функции. Его текст соответствует тексту объявленного прототипа этой функции. Далее, внутри фигурных скобок этой функции вписываются команды, обеспечивающие рисование на битмапе.

б) Задаем стиль и цвет линий, которыми будут прорисовываться оси:

FBC->Pen->Style=psSolid;         // Устанавливаем свойства пера

FBC->Pen->Color=clBlack;

Здесь - **FBC** это идентификатор макроса для выражения **formBitmap->Canvas**, который мы введем в целях уменьшения длины этой цепочки в командах прорисовки изображения и настройки параметров рисования. Поэтому, в область объявления макросов необходимо ввести следующую запись:

#define FBC formBitmap->Canvas

в) Проведем горизонтальную ось **X** с помощью команд:

FBC->MoveTo(0, H/2);              // Рисуем ось Х

FBC->LineTo(W, H/2);

Уберите заглушку, установленную перед вызовом **ShowOsi()** в функции **ShowAll()** и запустите проект на исполнение для проверки правильности прорисовки горизонтальной оси.

г) Проведем вертикальную ось **Y** с помощью этих же команд с координатами, соответствующими положению оси (выполнить самостоятельно). Запустите проект на исполнение для проверки.

д) Формируем стрелки на концах осей, используя команды **MoveTo** и **LineTo**(самостоятельно).

е) Устанавливаем буквы ‘**X**’ и ‘**Y**’ возле осей:

FBC->TextOut(x1, y1, 'X');

FBC->TextOut(x2, y2, 'Y');

Здесь **x1, y1, x2, y2** – координаты расположения символов **‘X’** и **‘Y’**.

Реальные значения **x1, y1, x2, y2**  надо задать заранее, перед вызовом команд TextOut() (самостоятельно). Переменные **x1, y1, x2,y2** объявите локально (внутри данной функции) как **int**.

ж) Прорисовываем отсчеты на осях  **X** и **Y** в виде коротких отрезков (количество отсчетов на каждой полуоси равно **nOs**), с использованием цикла **for**. Внутри этого же цикла вводим команду, выполняющую простановку соответствующих цифр (от 1 до значения **nOs**) возле каждого отсчета на полуоси. Пример простановки отсчетов и обозначений на положительной полуоси **OХ**имеет следующий вид:

h=W/(2\*(nOs+1));

hh=h;

for (int i=1; i<nOs+1; i++)

 { FBC->MoveTo((W/2+hh), (H/2-3));         //Отсчеты на полуоси +Х

     FBC->LineTo((W/2+hh), (H/2+3));

     // Здесь самостоятельно вписываем команды для остальных полуосей

    FBC->TextOut((W/2+hh), (H/2+10), i);     //Оцифровка полуоси +Х

    // Здесь самостоятельно вписываем команды для остальных оцифровок

    hh=h\*(i+1);

  } // цикл завершен

Здесь переменная  **h**– цена делений (отсчетов), на которые разбиваются оси**X**и**Y.**Она зависит от ширины окна и от количества отсчетов на осях, задаваемых переменной масштабирования осей**nOs.**   
      Переменная  **hh** – отстояние конкретного отсчета от центра системы координат.

Переменная  **nOs** – это количество отсчетов на любой из полуосей. Начальное значение этой переменной задано в функции **FormCreate()** в виде: **nOs** =10. В дальнейшем её значение можно изменять с помощью компонента **TrackBar1**.

Запустите программу на исполнение и убедитесь в правильности прорисовки осей.

**7.10. Разработка функции построения сетки ShowStk()**

В программный код файла реализации **Unit3.cpp** впишем заголовок этой функции:

void ShowStk(void)

Затем, внутри фигурных скобок этой функции необходимо вписать команды рисования пунктирных горизонтальных и вертикальных линий. Процесс построения сетки аналогичен прорисовке отсчетов на осях в функции **ShowOsi()**. Здесь также необходимо использовать оператор цикла для прорисовки параллельных пунктирных линий, совпадающих с отсчетами на осях. Только длина этих линий будет равна не длине отсчетов, а длине каждой из осей координат соответственно.

Перед началом прорисовки необходимо установить стиль линий - пунктирный и цвет линий – серый:

FBC->Pen->Style=psDot;      // Устанавливаем стиль линий для сетки - пунктир

FBC->Pen->Color=clSilver;    // Задаем цвет линий сетки – серый

  // Здесь самостоятельно вписываем команды прорисовки горизонтальных и

   // вертикальных линий сетки в цикле

      После окончания прорисовки сетки свойства пера необходимо вернуть в исходное состояние (см. п. 7.9).

Как видно из текста программного кода функции **ShowAll(),** прорисовка сетки осуществляется только в том случае, если включен флажок компонента **CheckBox1.**

Для того, чтобы при переключении состояния флажка выполнялась прорисовка сетки, или наоборот не выполнялась, необходимо создать обработчик события **CheckBox1Click**, с помощью которого можно вызывать команду **FormPaint(0)**  для перерисовки графического окна. При этом будет производиться вызов конвейера, с входящими в него командами проверки включения флажка и последующей прорисовки сетки. Порядок создания этого обработчика приведен в разделе 7.16.

Не забудьте убрать заглушку на вызове **ShowStk()** в функции **ShowAll()**.

После этого можно запустить программу на исполнение и убедиться в том, что сетка прорисовывается при включении флажка компонента “**Сетка**”, и наоборот, исчезает при его выключении.

**7.11. Масштабирование осей координат**

Принцип масштабирования осей координат и объектов, расположенных в графическом окне формы основан на изменении значения переменной  **nOs**, которая отвечает за количество отсчетов на координатной полуоси. Эта переменная в свою очередь влияет на цену деления **h** для осей **X** и  **Y,**  а следовательно и на конкретные размеры осей и объектов. Изменение значения переменной **nOs** в программе осуществляется с помощью компонента **TrackBar**. Для его использования создадим функцию обработки события на перемещение движка **TrackBar1**, для чего необходимо дважды щелкнуть по этому компоненту на поле конструктора формы **Form3**. В окне редактора кода **Unit3.cpp** появится заголовок новой функции, внутри фигурных скобок которой впишем:

void \_\_fastcall TForm3::TrackBar1Change(TObject \*Sender)

{

 nOs=Form3->TrackBar1->Position;     // Читаем переметры TrackBar1

 FormPaint(0);                                         // Запускаем перерисовку графич. окна

}

Здесь, как видно из текста функции, осуществляется присвоение переменной **nOs** значения позиции движка - **Position**. Затем вызывается команда **FormPaint(0)**  для перерисовки графического окна с другими размерами осей и фигуры.

Запустите программу на исполнение и убедитесь в том, что перемещение движка компонента **TrackBar1** вызывает изменение масштаба осей координат.

**7.12. Разработка функции прорисовки фигуры  ShowFig()**

Процесс прорисовки фигуры состоит из нескольких этапов. Сначала устанавливаем стиль пера и цвет фигуры. Затем преобразуем текущие отнормированные координаты фигуры в экранные. И только после этого выполняем собственно прорисовку: проводим линии от вершины **A** к вершине **B**, затем к вершине **C**, затем к вершине **D** и снова к **A**. После этого устанавливаем точки (маленькие окружности) в каждой из вершин фигуры. В заключение возвращаем свойствам пера исходные значения.

void ShowFig(TColorcol, float matrT[4][3])

{

 FBC->Pen->Color=col;                            // Устанавливаем цвет фигуры

 matrEkr[0][0]=W/2+int(h\*matrT[0][0]); // Переводим координаты фигуры в экранные

 matrEkr[0][1]=H/2-int(h\*matrT[0][1]);

 matrEkr[1][0]=W/2+int(h\*matrT[1][0]);

 matrEkr[1][1]=H/2-int(h\*matrT[1][1]);

//  и т.д., для всех восьми элементов матрицы matrEkr[4][2]

 FBC->MoveTo(matrEkr[0][0], matrEkr[0][1]);    // Устанавливаем перо в "А"

 FBC->LineTo(matrEkr[1][0], matrEkr[1][1]);       // Рисуем линию от "А" к "В"

 FBC->LineTo(matrEkr[2][0], matrEkr[2][1]);       // Рисуем линию от "В" к "С"...

// и т.д., для всех четырех вершин фигуры

 FBC->Ellipse(matrEkr[0][0]-2, matrEkr[0][1]-2,

                         matrEkr[0][0]+2, matrEkr[0][1]+2);  // Рисуем окружность в "А"

// и т.д., для всех четырех вершин фигуры

 FBC->Pen->Color=clBlack;            // Восстанавливаем исходные свойства пера

}

Снимаем заглушки с вызовов функций **ShowFig()** входящих в функцию **ShowAll()** и запускаем программу на исполнение. В графическом окне проекта должна прорисоваться исходная фигура в заданных исходных координатах с заданным цветом линий.

**7.13. Разработка функции прорисовки обозначений  Oboznath()**

Любая фигура, изображаемая на экране обладает, как минимум, двумя группами координат своих вершин. Первая группа – это её математические координаты, т.е. те, которые описывают местоположение фигуры в декартовой системе координат и хранятся в матрицах **matrIsx[4][3]** и **matrRez[4][3]** в виде вещественных чисел. Вторая группа координат – это экранные координаты фигуры, определяющие её физическое местоположение в графическом окне, и задаются в пикселях. Понятно, что математические координаты не совпадают с экранными, хотя они и описывают одно и то же место на экране. Поэтому, когда нам необходимо проставить буквенные обозначения с координатами возле каждой вершины мы должны придерживаться следующего правила: координаты возле вершин выводятся математические, а место, куда они проставляются, определяется в экранных координатах. Все эти особенности надо учитывать при заполнении списка фактических параметров для стандартной функции **TextOut()**, обеспечивающей вывод обозначений.

Пример вывода обозначения для вершины **А** фигуры в формате **А(x, y)** имеет следующий вид:

void Oboznath(float matrTo[4][3])

{

   FBC->TextOut(W/2+int(h\*matrTo[0][0]), H/2-int(h\*matrTo[0][1])-20,

                           "A("+FloatToStrF(matrTo[0][0], ffGeneral,2,1)+";"+

                            FloatToStrF(matrTo[0][1], ffGeneral,2,1)+")");

.//...и т.д., для остальных вершин: B, C, D

}

Как видно из текста программного кода функции **ShowAll(),** прорисовка обозначений осуществляется только в том случае, если включен флажок компонента **CheckBox2.** Для того, чтобы при переключении состояния флажка выполнялась прорисовка обозначений, или наоборот удалялась, необходимо создать обработчик события **CheckBox2Click,**с помощью которого можно вызывать команду **FormPaint(0)** для перерисовки графического окна. Порядок создания этого обработчика приведен в разделе 7.16.

Снимаем заглушки с вызовов функций **Oboznath()** входящих в функцию **ShowAll()** и запускаем программу на исполнение. Теперь можно убедиться в том, что обозначения прорисовываются возле каждой из вершин фигуры после включения флажка компонента “**Обозначения**” и наоборот, исчезают после его выключения.

**7.14. Создание обработчика события для кнопки "Запуск"**

Назначение обработчика события для кнопки "**Запуск**" (**Button1)** заключается в том, чтобы с его помощью запускать процессы преобразования и нормировки координат фигуры, а затем и прорисовки фигуры. Но, как уже говорилось выше, все эти действия запускаются не сразу, а через вызов функции перерисовки графического окна **FormPaint()**, внутри которой прописана функция графического конвейера **GrafKonv()**.

Учитывая то, что мы задались целью выполнять видовые преобразования фигуры в режиме анимации, нам необходимо в состав обработчика **Button1** ввести оператор цикла **For**, из которого мы будем вызывать функцию **FormPaint(0)** много раз, в соответствии с требованиями анимации.

Вообще, под анимацией понимается такое преобразование, при котором создается эффект плавного, непрерывного изменения положения фигуры или её размеров. Для этого все перемещение разбивается на множество маленьких перемещений, или шагов, и затем в цикле выполняют перерисовку фигуры столько раз, на сколько разбито полное перемещение (в нашем случае на 100 шагов). Чтобы не происходило суммирование ошибок вычисления, неизбежных при любых математических операциях, каждое пошаговое преобразование будем производить не относительно предыдущего положения фигуры, а относительно некоторого неизменного начального положения. Для этого у нас имеется матрица исходных координат фигуры **matrIsx[4][3]**. Далее, пересчитаем параметры матрицы преобразования**Tpr[3][3]** для одного шага анимации фигуры, т.е для 1/100 части от заданного отрезка перемещения. После этого можно вызывать функции **Preobr(), Normir()** и **ShowAll()**, входящие в состав графического конвейера, для перерисовки фигуры в цикле.

Порядок создания функции обработки события **Click** для левой кнопки мыши аналогичен действиям, описанным в лаб. раб. №1. Необходимо дважды щелкнуть по кнопке "**Пуск**" (**Button1)** на конструкторе формы **Form3**. В окне редактора кода **Unit3.cpp**появится заголовок новой функции, внутри которой необходимо вписать команды, выполняющие следующие действия:

- чтение значений параметров видового преобразования из ячеек компонента **StringGrid2**;

- ввод этих значений в матрицу преобразования **Tpr[3][3]**;

- формирование анимационного цикла, в котором выполняются следующие действия:

        - пересчет значений параметров матрицы преобразования **Tpr[3][3]**  для их 1/100 части;

        - вызов перерисовки графического окна, для запуска графического конвейера;

        - вызов функции временно́й задержки **Sleep(n)**;

- вывод значений матрицы координат преобразованной фигуры в ячейки компонента **StringGrid3**.

С учетом вышесказанного, текст обработчика события для кнопки "**Запуск**" будет иметь вид:

void \_\_fastcall TForm3::Button1Click(TObject \*Sender)

{

 a=StrToFloat(Form3->StringGrid2->Cells[0][0]);

 b=StrToFloat(Form3->StringGrid2->Cells[1][0]);

 p=StrToFloat(Form3->StringGrid2->Cells[2][0]);

 d=StrToFloat(Form3->StringGrid2->Cells[0][1]);

 e=StrToFloat(Form3->StringGrid2->Cells[1][1]);

 q=StrToFloat(Form3->StringGrid2->Cells[2][1]);

 l=StrToFloat(Form3->StringGrid2->Cells[0][2]);

 m=StrToFloat(Form3->StringGrid2->Cells[1][2]);

 s=StrToFloat(Form3->StringGrid2->Cells[2][2]);

 Tpr[0][0]=a; Tpr[0][1]=b; Tpr[0][2]=p;

 Tpr[1][0]=d; Tpr[1][1]=e; Tpr[1][2]=q;

 Tpr[2][0]=l; Tpr[2][1]=m; Tpr[2][2]=s;

 if(Form3->CheckBox3->Checked)

   for(float i=0; i<100; i++)

   {Tpr[0][0]=1+(((a-1)\*(i+1))/100);

    Tpr[1][1]=1+(((e-1)\*(i+1))/100);

    Tpr[2][0]=(i+1)\*(l/100);

    Tpr[2][1]=(i+1)\*(m/100);

    Tpr[2][2]=1+(((s-1)\*(i+1))/100);

    FormPaint(0);

    Sleep(10);

   }

 else FormPaint(0);

 StringGrid3->Cells[0][0]=matrRez[0][0];

 StringGrid3->Cells[1][0]=matrRez[0][1];

 StringGrid3->Cells[2][0]=matrRez[0][2];

 StringGrid3->Cells[0][1]=matrRez[1][0];

 StringGrid3->Cells[1][1]=matrRez[1][1];

 StringGrid3->Cells[2][1]=matrRez[1][2];

 //ит.д., для остальных шести ячеек матрицы  matrRez[4][3];

}

Как видно из текста этой программы, она позволяет выполнять анимацию только для переноса, масштабирования и отображения. Для осуществления анимации сдвига и перспективного проецирования необходимо разрабатывать гораздо более сложную функцию, так как там требуется пересчитывать каждую вершину по отдельности с использованием конвейера обработки изображения, и здесь мы это рассматривать не будем.

Сохраните и запустите проект на исполнение.

На рабочем поле открывшегося окна проекта, в ячейках компонента **StringGrid2,** установите параметры преобразования и щелкните мышью по кнопке "**Пуск**". В графическом окне, рядом с исходной фигурой **ABCD**, должна появиться вторая, преобразованная фигура. Местоположение и форма новой фигуры должны соответствовать заданным параметрам преобразования.

Если перед нажатием на кнопку "**Пуск**" включить флажок компонента "**Анимация**", то преобразование фигуры должно происходить плавно, в замедленном режиме.

**7.15. Создание обработчика события для кнопки "Исходное"**

Обработчик события кнопки "**Исходное**" (**Button2)**  предназначен для следующих действий:

- восстановления исходных значений матрицы видового преобразования **Tpr[3][3]**;

- очистки всех ячеек компонентов **StringGrid2**  и **StringGrid3**;

- запуска функции перерисовки графического окна.

Тогда текст его программного кода будет иметь вид:

void \_\_fastcall TForm3::Button2Click(TObject \*Sender)

{

 Tpr[0][0]=1; Tpr[0][1]=0; Tpr[0][2]=0;

 Tpr[1][0]=0; Tpr[1][1]=1; Tpr[1][2]=0;

 Tpr[2][0]=0; Tpr[2][1]=0; Tpr[2][2]=1;

 StringGrid2->Cells[0][0]=Tpr[0][0];

 StringGrid2->Cells[1][0]=Tpr[0][1];

 StringGrid2->Cells[2][0]=Tpr[0][2];

 // и т.д., для остальных шести ячеек компонента StringGrid2

 StringGrid3->Cells[0][0]=' ';

 StringGrid3->Cells[1][0]=' ';

 StringGrid3->Cells[2][0]=' ';

 StringGrid3->Cells[0][1]=' ';

 StringGrid3->Cells[1][1]=' ';

 StringGrid3->Cells[2][1]=' ';

 // и т.д., для остальных шести ячеек компонента StringGrid3

 FormPaint(0);

}

Здесь заполнение ячеек компонентов **StringGrid** конечно можно выполнить в цикле, что более предпочтительно для тех, кто считает себя настоящим программистом.

**7.16. Создание обработчиков событий  "Сетка" и "Обозначения"**

Необходимость создания обработчиков событий для этих компонентов заключается в том, чтобы вызывать перерисовку всего изображения в случае манипуляции пользователя с этими компонентами. При этом должен происходить перезапуск функции**FormPaint(TObject \*Sender)** с помощью команды **FormPaint(0)**.

Для создания обработчика для флажка "**Сетка**" дважды щелкните по объекту **CheckBox1** на конструкторе формы **Form3**. В окне редактора кода **Unit3.cpp** появится заголовок этой функции, внутри которой можно вписать команду вызова перерисовки графического окна:

void \_\_fastcall TForm3::CheckBox1Click(TObject \*Sender)

{

 FormPaint(0);

}

Аналогичный обработчик создайте для флажка "**Обозначения**" (**CheckBox2)**.

На этом разработка файла реализации для лабораторной работы №2 заканчивается. Сохраните проект и запустите на исполнение для проверки его работоспособности при выполнении видовых преобразований над фигурой.